Coursera Deep Learning Specialization Notes: Structuring Machine Learning Projects

Amir Masoud Sefidian

Version 1.0, February 2023

Contents

Preface

A couple of years ago I completed Deep Learning Specialization taught by AI pioneer Andrew Ng. I found this series of courses immensely helpful in my learning journey of deep learning. After years, I decided to prepare this document to share some of the notes which highlight key concepts I learned in the third course of this specialization, Structuring Machine Learning Projects. This course is all about how to build ML projects, get results quickly and iterate to improve these results. It gives delightful insights into how to diagnose the outcomes of our models so that we can see where the performance problem is coming from if there is one: small training set, different distributions of train and test set, over-fitting, and other problems are covered, along with their solutions.

The content of this document is mainly adapted from this GitHub repository. I have added some explanations, illustrations, and visualization to make some complex concepts easier to grasp for readers. This document could be a good reference for Machine Learning Engineers, Deep Learning Engineers, and Data Scientists to refresh their minds on the fundamentals of deep learning. Please don't hesitate to contact me via my website (Sefidian Academy) if you have any questions.

Happy Learning!
Amir Masoud Sefidian

1 Structuring machine learning projects

1.1 Orthogonalization

- Separating the hyperparameter tuning into different "dimensions". "Using different knobs" for each objective/tuning/assumption in ML:
  - Fit training set well on the cost function. Knobs to improve: Try out bigger NN, Try out different NN architecture, Try out a different optimizer (Momentum, RMSprop, Adam, $\cdots$), and train longer (more iterations).
  - Fit dev set well on the cost function. Knobs to improve: Regularization, Bigger training set.
  - Fit test set well on the cost function. Knobs to improve: Bigger dev set.
  - Perform well on real-world data: Knobs to improve: Change dev set (different probability distribution), Change cost function.
- Orthogonalization states that a specific action should only affect one of the above steps, not several at the same point.
- A bad example of a "knob" is early stopping because it simultaneously affects the fit of the training set and the dev set, therefore is not a targeted "knob".

1.2 Single number evaluation metric

There is usually a trade-off between:

- Precision: the percentage of images identified as cats that actually are cats:

$$P = \frac{true\_positives}{true\_positives + false\_positives}$$

- Recall: the percentage of cats correctly classified out of all cat images:

$$R = \frac{true\_positives}{true\_positives + false\_negatives}$$

Combine the 2 metrics into a single number with the F_1 Score:

- Average of precision P and recall R:

$$F_1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}, \text{ this is the harmonic mean of P and R}$$

1.3 Satisfying and Optimization metric

- Satisfying metric: one that only needs to be *acceptable* value (e.g. running time must be $\leq$ 100ms).
- Optimizing metric: one that we want to optimize (e.g. accuracy).

1.4 Training/dev/test sets Distributions

- Mismatched training and dev/test sets are often rooted in the nature of Deep Learning, which usually requires a lot of labeled data.
- Dev and Test sets must (or at least should) come from the same distribution. Dev and test set should reflect the data that you expect to get in the future and consider it important to do well on.
  - The traditional splits (70/30 or 60/20/20) are no longer useful if you have large datasets (e.g. 1 million records). In the case of deep learning with huge datasets, use a 98/1/1 split.
  - Not having a test set might be OK (sometimes there is even only a training set, though that is not recommended).
- Metrics that evaluate classifiers should reflect what is expected from a classifier, for example, measuring an algorithm just by classification error does not take into account the fact that some of the false positives that it classifies are actually porn images instead of cats, in which case, a revised (e.g. weighted version) the metric should be used instead.
  - Use orthogonalization, first place the target (the right metric), and only then try to improve the performance according to that metric.
- If doing well on your metric + dev/test set does not correspond to doing well on your application, change your metric and/or dev/test set.
- Do NOT run for too long with an appropriate dev set or applicable metric.

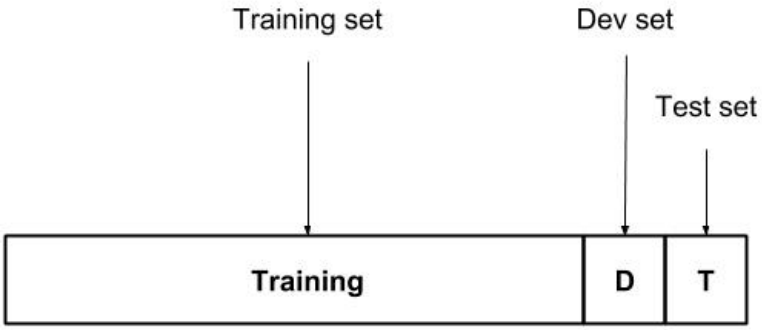

Figure 1: Train/Dev/Test Sets

1.5 Comparing to human-level performance

- Performance of an algorithm can surpass **human-level performance**, but even then it will be bound by the **Bayes optimal error** (or **Bayes error**), which is the best possible error value that can be achieved.
- There are two reasons why performance slows down when human-level performance is surpassed because humans are already close to "Bayes optimal error" in some cases.
- So long as ML is worse than humans, you can:

- Get labeled data from humans.
- Gain insight from manual error analysis.
- Better analysis of bias/variance.

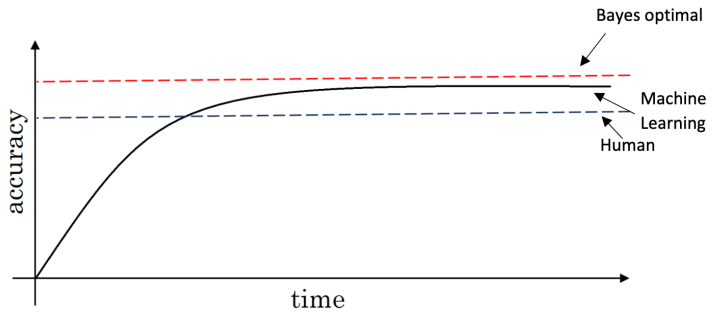

Figure 2: Human Performance

Avoidable bias

- Focus on improving bias: if human performance is much better than performance on the training set.
- Focus on improving variance: if human performance is close to performance on the training set, but performance on the dev set is significantly lower than training performance.
- The difference between training error and human error is **avoidable bias**.
- See Figure (4).

  Scenario A: There is a 7% gap between the performance of the training set and the human-level error. It means that the algorithm isn't fitting well with the training set since the target is around 1%. To resolve the issue, we use bias reduction techniques such as training a bigger neural network or running the training set longer.

  Scenario B: The training set is doing good since there is only a 0.5% difference with the human-level error. The difference between the training set and the human-level error is called avoidable bias. The focus here is to reduce the variance since the difference between the training error and the development error is 2%. To resolve the issue, we use variance reduction techniques such as regularization or collecting a bigger training set.

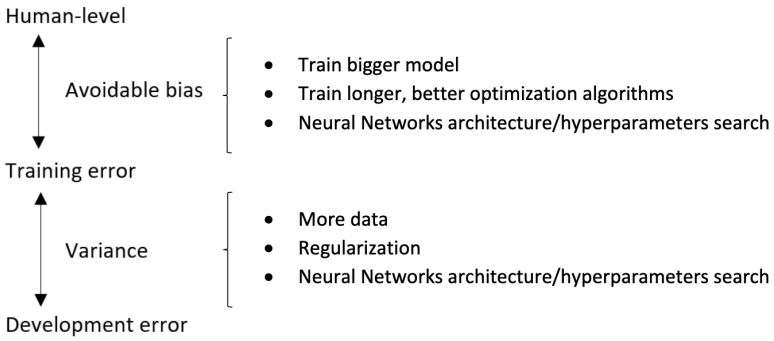

Figure 3: Avoidable Bias and Variance

	Classification error (%)	
	Scenario A	Scenario B
Humans	1	7.5
Training error	8	8
Development error	10	10

Figure 4

Understanding human-level performance

- **Human-level** error as a proxy for **Bayes error**: we use the best possible human-level error when trying to approximate Bayes error. This matters when the avoidable bias and the variance are already low.

Exceeding human-level performance

- If an algorithm surpasses human-level performance it is not clear what the avoidable bias is, and therefore we don't know if we should focus on improving bias or variance. Examples are:
  - Online advertising, Product recommendations, Logistics, and Loan approval: all structured data problems, NOT natural perception problems.
  - Speech recognition, some image recognition, medical ECG and cancer analysis, etc.

Improving your model performance

- Improve **avoidable bias**:
  - Train bigger model
  - Train longer/better optimization algorithms (momentum, RMSProp, Adam)
  - NN architecture/hyperparameters search (RNN, CNN, etc.)
- Improve variance:
  - More data
  - Regularization (L2, dropout, data augmentation)
  - NN architecture/hyperparameters search (RNN, CNN, etc.)

1.6 Error analysis

- Finding the most promising way to reduce error, by manually going through examples and finding out what the misclassified examples refer to (e.g. dogs, when trying to classify cats).
  - Depending on the percentage of these cases, choose the most prominent cases to focus on. If there are multiple examples, each team can focus on a different case. The conclusion of this process gives you an estimate of how worthwhile it might be to work on each of these different categories of errors. For example, clearly in Figure (5), a lot of the mistakes we made on blurry images, and quite a lot on were made on great cat images.
- Incorrectly labeled examples:

Error analysis

Image	Dog	Great Cat	Blurry	Incorrectly labeled	Comments
...					
98				✓	Labeler missed cat in background
99		✓			
100				✓	Drawing of a cat; Not a real cat.
% of total	8%	43%	61%	6%	

Figure 5: Error Analysis.

  - Deep Learning is robust to random errors in the training set - not worth fixing manually. But we should be cautious about systematic errors.
  - The frequency of mislabeled examples due to incorrect labeling should be calculated similarly to other mislabeled example analyses mentioned above.

- Correcting incorrect dev/test set examples
  - Whatever is done to the dev set we should also do to the test set so that they continue to come from the same distribution. The training set might come from a slightly different distribution.
  - Consider examining the (mislabeled) examples that the algorithm got right, not just those that it misclassified.

1.7 Build your first system quickly, then iterate

- Includes setting up dev/test set and metric.
- Allows you to identify areas where the algorithm is not performing well.
- Use Bias/Variance analysis and error analysis to prioritize the next steps.

1.8 Training and testing on different distributions

Suppose we have 200k examples from distribution A (high-resolution cat images from web pages) and only 10k images from another distribution B (lower-resolution cat images from a mobile app). It also happens that the images that we will need to classify are from distribution B.

- OPTION 1 (BAD): Mix A and B, randomized the images, and use a training/dev/test split of 205k/ 2.5k/2.5k images. This is bad because distribution B is the target, and there will be only a low number of the dev/test set in this case.
- OPTION 2 (GOOD): Use 200k from A as training examples and split the 10k from B into dev and test sets. This is much better because dev and test are used to set the "target" distribution (B in this case).

Variance (difference between error in the training and dev sets) might be higher (assuming OPTION 2) due to the different distributions of the two sets. A **training-dev** set with the same distribution as that of the *training set* (A) but *it is not used for training*. It can be created to determine if:

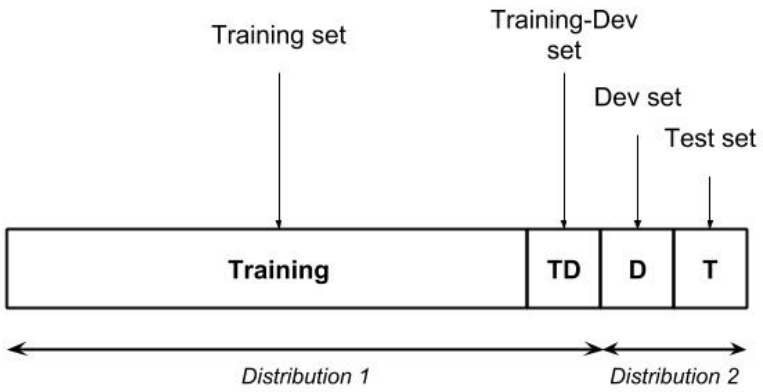

Figure 6: Training-Dev Set

- The error is due to overfitting the training set (variance) when the training-dev error is close to the dev error,
- or if it is due to the model not generalizing well to distribution B (**data mismatch** problem) if the training-dev error is close to the training error.

Note the 4 types of errors (Figure (7)):

- avoidable bias: human_level - training_set_error
- variance: training_set_error - training_dev_set_error
- data mismatch: training_dev_set_error - dev_error
- degree of overfitting to dev: dev_error - test_error

Bias and Variance Analysis Example (Figure (8)):

Scenario A:

If the development data and training set come from the *same distribution*, then there is a large variance problem and the algorithm is not generalizing well from the training set. However, when the training data and the development data come from *different distributions*, this conclusion cannot be drawn. There isn't necessarily a variance problem. The problem might be that the development set contains images that are more difficult to classify accurately. When the training set, development, and test sets distributions are different, two things change at the same time. 1) the algorithm trained in the training set but not in the development set. 2) The distribution of data in the development set is different. It's difficult to know which of these two changes produces this 9% increase in error between the training set and the development set.

Scenario B:

The error between the training set and the train-dev set is 8%. In this case, since the training set and train-dev set come from the same distribution, the only difference between them is the model sorted the data in the training and not in the training development. The model is not generalizing well to data from the same distribution that it hadn't seen before. Therefore, we have really a variance problem.

Scenario C: In this case, we have a mismatch data problem since the two datasets come from different distributions.

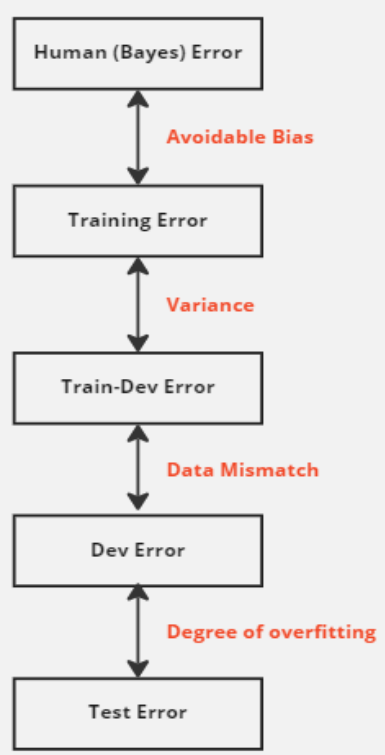

Figure 7: Types of Error.

	Classification error (%)					
	Scenario A	Scenario B	Scenario C	Scenario D	Scenario E	Scenario F
Human (proxy for Bayes error)	0	0	0	0	0	4
Training error	1	1	1	10	10	7
Training-development error	-	9	1.5	11	11	10
Development error	10	10	10	12	20	6
Test error	-	-	-	-	-	6

Figure 8: Bias and Variance Analysis Example

Scenario D:

In this case, the avoidable bias is high since the difference between Bayes error and training error is 10%.

Scenario E:

In this case, there are two problems. The first one is that the avoidable bias is high since the difference between Bayes error and training error is 10% and the second one is a data mismatched problem.

Scenario F:

Development should never be done on the test set. However, the difference between the development set and the test set gives the degree of overfitting to the development set.

Addressing data mismatch

- Carry out manual error analysis and try to understand the differences between the training and dev/test sets.
- Make training data more similar (**artificial data synthesis**), or collect more data similar to

dev/test sets. For example, if you find that car noise in the background is a major source of error, one thing you could do is simulate noisy in-car data. Make sure that artificial data synthesis is varied enough to avoid overfitting to one dimension or a sub-set domain of the synthesized data.

1.9 Transfer learning

Includes re-using a previously trained model (e.g. in general image recognition if your task is to make radiology diagnosis). This usually requires replacing the last layer of the classifier with a new layer (or layers) with weights and biases randomly initialized. Then there are two options:

- Retrain only the last layer if we don't have enough data in the new (e.g. radiology) dataset.
- Or, if there is sufficient data, retrain the entire model, in which case:
  - The initial phase of training (e.g. on image recognition) is called **pre-training**.
  - Training on the new dataset (e.g. radiology) is called **fine-tuning**.
- Transfer learning (from models A to B) makes sense if:
  - Task A and B have the same input X.
  - You have a lot more data for Task A (e.g. image recognition) than Task B (e.g. radiology).
  - Low-level features from A could be helpful for learning B.

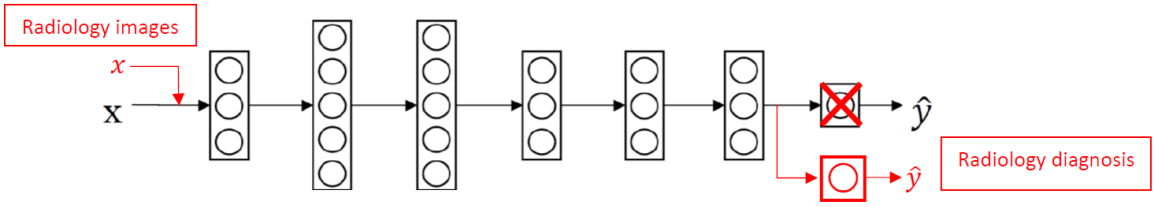

Figure 9: Transfer Learning

You can find more detailed guidelines about transfer learning on my website:
Guidelines to use Transfer Learning in Convolutional Neural Networks

1.10 Multi-task learning

Learning to perform several tasks simultaneously. For instance, learning to identify multiple objects at the same time (e.g. automatic car driving). The loss function must be a vector $\hat{y}^{(i)}$ where each entry corresponds to one object. One cost function for this can be:

$$J = \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{c}\mathcal{L}(\hat{y}_j^{(i)}, y_j^{(i)})$$

Where c is the number of object classes/types and $\mathcal{L}$ is the logistic loss function.

- Unlike Softmax, one example (e.g. image) can have multiple labels.
- $\hat{y}$ can "unknown" values in certain positions. To calculate $\mathcal{L}$ in that case, we can simply exclude those terms from the inner sum of the cost function.

Multi-task learning makes sense when:

- Training on a set of tasks that could benefit from having shared lower-level features.
- Usually, the amount of data you have for each task is quite similar. If we focus on a single target task, then we need to make sure that the other tasks have a sufficiently large number of examples.
- Can train a big enough neural network to do well on all tasks with better performance (e.g. computer vision or object detection).

Transfer learning is much more used currently in practice.

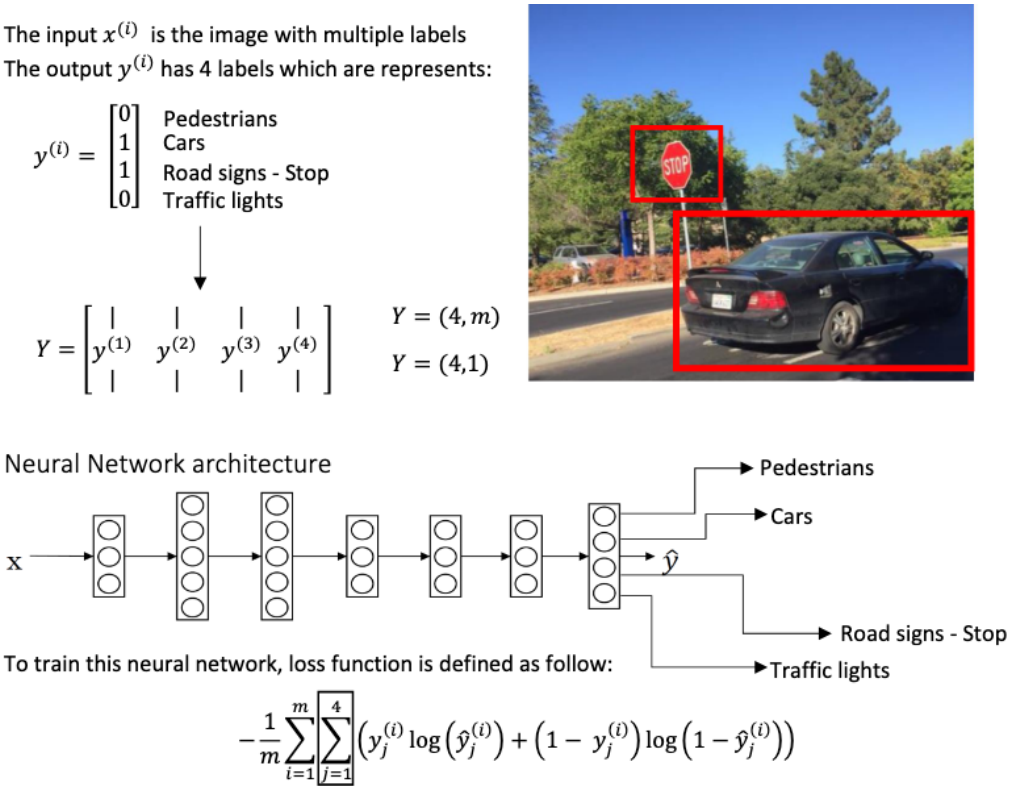

Figure 10: Multi-Task Learning

1.11 End-to-end learning

- A single classifier can replace multiple stages of processing in previous approaches (e.g. speech processing).
- For small datasets, the traditional pipeline approach works better and for large amounts of data neural networks work better.

Pros of end-to-end learning:

- Let the data speak (features are learned, not forced).
- Less hand-designing of components is needed.

Cons of end-to-end learning:

- May need a large amount of data.
- Excludes potentially useful hand-designed components.

Key question: Do you have sufficient data to learn a function of the complexity needed to map x to y?

If not: It is also possible to use Deep Learning (DL) to learn individual components. Carefully chose $X \to Y$ to learn depending on what tasks you can get data for.

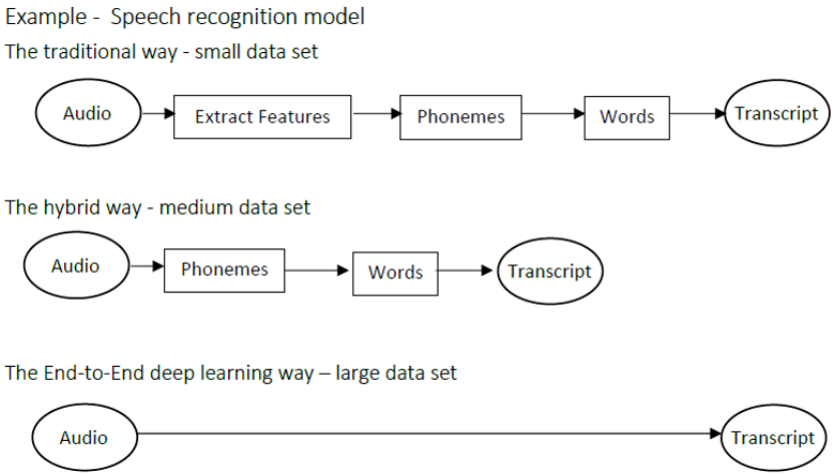

Figure 11: End-to-End Learning

References

[1] S. Zivkovic. Language Modelling and Sampling Novel Sequences. https://datahacker.rs/004-rnn-language-modelling-and-sampling-novel-sequences/.